# Project Proposal

Dane Fichter and Jon Cronin

October 4, 2013

## 1 Abstract

Modern smartphones are responsible for a growing number of functions in our daily lives, and, consequently, hold an increasingly large amount of sensitive data. Many mobile operating systems alert the user to each application's use of sensitive data upon install, but cannot prevent applications from using this data in malicious ways. The research community is aware of this threat, and has developed software to detect these misuses of private data. However, this software is not foolproof. A clever attacker can leak private data from an application in an undetected manner using techniques such as covert channels or implicit flow. Our research will test these two techniques. We will determine which of these most efficiently and reliably evades the program TaintDroid, which is software designed to prevent the leak of sensitive data from Android applications.

## 2 Motivation

The issue motivating our project is related to smartphone security, in particular relating to potentially malicious applications abusing Androids permission based mechanisms. Smartphones allow users to download applications created by third party developers who present a potential risk. Particularly in the Android Marketplace, where applications are not thoroughly vetted for malicious intent, it is increasingly likely that apps are misusing sensitive information. Upon downloading an application, the user is prompted as to whether they would like to grant the app certain permissions, for example use of the devices microphone or the users contact list. This layer of security is deceivingly reassuring for many users, and multiple papers have been written detailing how these permissions can be (and are in practice) circumvented to access and relay sensitive information to external servers.

Our project is thus motivated by the existing research (see below) related to covert channels in the Android OS and detection of malicious applications, as well as real life security concerns we hope to highlight related to Android applications.

## 3 Background

Much work has been done related to malicious applications ability to extract sensitive data from a smartphone. Some of these techniques are very subtle and work by reading information into seemingly innocuous data. Two papers show how to use accelerometer data to infer keyboard strokes and pin entries simply from the motion of the phone. [1, 2] These applications that send the accelerometer data have permission to do just this, however the user is not expecting the data to be used in such a compromising way.

Similarly, Gasior and Yang show that one way to send this sensitive data to an outside party without being detected is to mask the transfer of this data with a steady stream of unrelated, non sensitive data to a server. [4] Specifically, they send a live video stream from the smartphone to the server, a process the user had approved, but they send each frame with a certain delay, long or short. These delays are used to encode a list of contacts in binary form, with two short delays representing a zero and two long delays representing a one. The smartphone does not detect any unapproved data being sent anywhere, but the contact list is sent nevertheless.

Marforio, Francillon, and Capkun present a different way to send sensitive data: an Application Collusion Attack. [6] In essence, they launch two seemingly unrelated applications, one of which has permission to access sensitive data, one of which has permission to access the network. They use a covert channel to then transmit the sensitive data from the app that has that permission to the app that can access the network, which then sends it to the external server. Like the idea given in the above paper, the Android OS does not detect this breach of security. Furthermore, the authors argue that they can evade extra detection software, such as TaintDroid.

TaintDroid attempts to follow the flow of sensitive data by giving it a special tag, called a taint tag, which allows it to see where and how this data is being used.[3] The creators of TaintDroid argue that it can detect many instances of personal information being misused by even relatively popular applications. However, they concede that TaintDroid has limitations in terms of how much wrongdoing it can detect. In particular, they write that TaintDroid is ineffective at detecting implicit flow, a flaw we hope to take advantage of in our project.

Another application that builds off TaintDroid is AppFence. AppFence uses TaintDroid as a base, but also supplies shadow data to applications it suspects are attempting to extract sensitive information.[5] This shadow data can be a dummy variable or an empty file, and its purpose is to prevent the users real data from being compromised.

## 4 Idea

Our goal is to test the effectiveness of various implementations of covert channels at accurately transmitting sensitive data while also avoiding detection by the Android OS as well as software such as TaintDroid. There are several well-documented methods by which data can be covertly transmitted from an application to a server. We will investigate these methods, as well as develop techniques of our own design, ultimately selecting a method which we will implement.

The authors of TaintDroid acknowledged a specific weakness of their program, a practice referred to as ”implicit flow”. This method is a succinct and lightweight way to deceive taint-tracking software. Since its implementation is relatively simplistic, we will also transmit sensitive data using this method.

Once both the covert channel and implicit flow approaches are successfully implemented, we will modify them until they successfully evade detection by TaintDroid. As soon as we have ensured that our software can reliably evade detection by TaintDroid, we will exhaustively test these applications for accuracy and efficiency.

## 5 Goals

**Milestone 1**

By our first milestone meeting, we hope to have implemented a functional Android application which uses some method of covert channel to transmit sensitive data. Furthermore, we hope to have configured TaintDroid to run with our application.

**Milestone 2**

By our second milestone meeting, we hope to have implemented a functional Android application which uses implicit flow to transmit sensitive data. We also hope to have made significant progress in developing our server-side testing framework.

**Milestone 3**

At this point, both of our covert channels and our testing framework must be completed. From Milestone 3 onward, our focus will be on exhaustively testing our software, compiling results about its efficiency, and devising ways to avoid detection from TaintDroid. Ultimately these results will be reported in a paper.

## References

[1] Adam J Aviv, Michael E Locasto, Shaya Potter, and Angelos D Keromytis. Ssares: Secure searchable automated remote email storage. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 129–139. IEEE, 2007.

[2] Liang Cai and Hao Chen. Touchlogger: inferring keystrokes on touch screen from smartphone motion. In *Proceedings of the 6th USENIX conference on Hot topics in security*, pages 9–9. USENIX Association, 2011.

[3] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI*, volume 10, pages 255–270, 2010.

[4] Wade Gasior and Li Yang. Network covert channels on the android platform. In *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '11, pages 61:1–61:1, New York, NY, USA, 2011. ACM.

[5] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These arent the droids youre looking for. *Retrofitting Android to Protect Data from Imperious Applications. In: CCS*, 2011.

[6] Claudio Marforio, Aurélien Francillon, Srdjan Capkun, Srdjan Capkun, and Srdjan Capkun. *Application collusion attack on the permission-based security model and its implications for modern smartphone systems.* Department of Computer Science, ETH Zurich, 2011.

[7] Clemens Orthacker, Peter Teufl, Stefan Kraxberger, Günther Lackner, Michael Gissing, Alexander Marsalek, Johannes Leibetseder, and Oliver Prevenhueber. Android security permissions–can we trust them? In *Security and Privacy in Mobile Information and Communication Systems*, pages 40–51. Springer, 2012.